

Research - Health Ledger Subsystem Design: Parallel Cryptographic Monitoring for System Diagnostics

Alpasan, Lois-Kleinner

2026

Abstract

Production AI systems require continuous health monitoring that generates diagnostic telemetry about system state, resource utilization, and operational anomalies. When this health telemetry must be preserved with cryptographic integrity for regulatory compliance and post-incident forensics, a dedicated health ledger subsystem provides separation of concerns from the primary audit ledger. This paper presents the design and analysis of the AIOSS Health Ledger Subsystem, a parallel cryptographic ledger format (AIOSH, magic byte 0x41494F5348) specifically optimized for high-frequency system diagnostics. The health ledger records structured metric entries (CPU, memory, disk, network, AI model metrics) with millisecond-precision timestamps and SHA3-256 hash chain integrity. A companion manifest file provides summary statistics and cross-references to the primary AIOSS audit ledger for correlated forensics. We evaluate the subsystem's overhead on instrumented systems, demonstrating less than 0.5% CPU utilization overhead and 2.1 MB/hour storage consumption at 1 Hz polling frequency. The health ledger supports configurable retention policies, automatic pruning of metrics exceeding retention windows, and integrity verification that operates independently of the primary ledger. We analyze the security properties of the parallel health ledger design, including the manifest cross-signing protocol that links health and audit ledgers. The findings establish that dedicated health ledgers p

Health Ledger Subsystem Design: Parallel Cryptographic Monitoring for System Diagnostics

Document ID: AIOSS-RES-010-001 **Version:** 1.0.0 **Date:** 2026-06-20 **Classification:** Academic Research

Abstract

Production AI systems require continuous health monitoring that generates diagnostic telemetry about system state, resource utilization, and operational anomalies. When this health telemetry must be preserved with cryptographic integrity for regulatory compliance and post-incident forensics, a dedicated health ledger subsystem provides separation of concerns from the primary audit ledger. This paper presents the design and analysis of the AIOSS Health Ledger Subsystem, a parallel cryptographic ledger format (AIOSH, magic byte 0x41494F5348) specifically optimized for high-frequency system diagnostics. The health ledger records structured metric entries (CPU, memory, disk, network, AI model metrics) with millisecond-precision timestamps and SHA3-256

hash chain integrity. A companion manifest file provides summary statistics and cross-references to the primary AIOSS audit ledger for correlated forensics. We evaluate the subsystem’s overhead on instrumented systems, demonstrating less than 0.5% CPU utilization overhead and 2.1 MB/hour storage consumption at 1 Hz polling frequency. The health ledger supports configurable retention policies, automatic pruning of metrics exceeding retention windows, and integrity verification that operates independently of the primary ledger. We analyze the security properties of the parallel health ledger design, including the manifest cross-signing protocol that links health and audit ledgers. The findings establish that dedicated health ledgers provide a practical foundation for integrity-preserving system monitoring in regulated AI deployments.

1. Introduction

System health monitoring is a critical operational requirement for production AI deployments. Metrics such as CPU utilization, memory consumption, GPU temperature, inference latency, and error rates provide essential visibility into system behavior and performance degradation. For regulated AI systems (EU AI Act high-risk classification, HIPAA-covered healthcare AI, FedRAMP-authorized government AI), health monitoring data must be preserved as audit evidence.

Two architectural approaches exist for health ledger storage: (1) integrating health metrics as entries within the primary audit ledger, and (2) maintaining a separate, parallel health ledger with cross-references. The AIOSS Health Ledger Subsystem adopts the parallel approach for several reasons:

1. **Frequency separation:** Health metrics at 1 Hz or higher would overwhelm the primary ledger, which records discrete audit events at much lower frequency
2. **Retention independence:** Health data may have shorter retention requirements than audit records
3. **Operational independence:** Health monitoring must continue even when the primary audit ledger is closed or finalized
4. **Performance isolation:** Health ingestion must not impact audit ledger append latency

The AIOSSH format (AI Open Signed Health) extends AIOSS with a dedicated magic byte and metric-optimized entry structure. This paper presents the design, implementation, and evaluation of the health ledger subsystem.

2. Literature Review

2.1 System Health Monitoring

Health monitoring of production systems has evolved from simple SNMP polling (Case et al., 1990) to sophisticated observability platforms. The Prometheus monitoring system (Turnbull, 2018) introduced a pull-based metrics collection model with a multi-dimensional data model. The OpenTelemetry standard (CNCF, 2024) unified metrics, traces, and logs collection.

For AI-specific monitoring, Google’s TensorFlow Extended (TFX) pipeline monitoring (Baylor et al., 2017) and Uber’s Michelangelo (Hermann & Del Balso, 2017) demonstrated the need for continuous model performance tracking. MLflow (Zaharia et al., 2018) provides experiment tracking but lacks operational health monitoring capabilities.

2.2 Cryptographic Integrity for Metrics

Most monitoring systems lack cryptographic integrity for stored metrics. Security monitoring (SIEM) systems provide some integrity through access control but not cryptographic hash chains. The work on tamper-evident logging (Crosby & Wallach, 2009; Pullkis et al., 2020) focused on discrete audit events rather than continuous metrics streams.

Secure audit logging for industrial control systems (Hadiosmanović et al., 2015) addressed integrity for SCADA system logs but with lower frequency requirements than AI health monitoring.

2.3 Health Ledger Concepts in Other Domains

Blockchain-based health monitoring has been proposed for IoT systems (Dorri et al., 2017) and healthcare (Yue et al., 2016). These designs focus on distributed consensus rather than local integrity verification. Apple’s Transparency, Consent, and Control (TCC) framework provides system-level audit logging on macOS but without cryptographic integrity.

The Linux Audit subsystem (Grubb, 2004) provides comprehensive system call auditing but generates unstructured text logs without hash chain integrity. The AIOSH health ledger format provides a structured, integrity-preserving alternative for critical system monitoring.

3. Technical Analysis

3.1 AIOSH Format Specification

The AIOSH (AI Open Signed Health) format uses a distinct magic byte and optimized entry structure:

```
Magic: 0x41 0x49 0x4F 0x53 0x48 0x0A 0x0D 0x0A
        A    I    O    S    H    \n    \r    \n
```

Health entry header (64 bytes fixed + variable payload):

Byte offset	Field	Type	Size	
0	entry_type	u8	1	
1	metric_group	u8	1	// 0=system, 1=AI model, 2=network
2	severity	u8	1	// 0=info, 1=warn, 2=error, 3=critical
3	flags	u8	1	
4	timestamp	u64	8	// Unix nanoseconds
12	metric_count	u16	2	// Number of key-value pairs
14	parent_hash	[32]u8	32	
46	reserved	[18]u8	18	

= 64 bytes header

Metric payload format (space-efficient binary encoding):

```
metric_entry = {
    metric_id: u16,           // Pre-registered metric identifier
    value_type: u8,          // 0=u64, 1=f64, 2=string
    value: [variable]        // 8 bytes (u64/f64) or variable (string)
}
```

3.2 Health Metric Taxonomy

Pre-registered metric IDs (first 64 reserved for system metrics):

ID	Metric	Type	Unit	Frequency
1	cpu_percent	f64	%	1 Hz
2	memory_used_bytes	u64	bytes	1 Hz
3	memory_total_bytes	u64	bytes	0.1 Hz
4	disk_read_bytes	u64	bytes/s	0.1 Hz
5	disk_write_bytes	u64	bytes/s	0.1 Hz
6	net_rx_bytes	u64	bytes/s	1 Hz
7	net_tx_bytes	u64	bytes/s	1 Hz
8	gpu_util_percent	f64	%	1 Hz
9	gpu_temp_celsius	f64	°C	0.5 Hz
10	inference_latency	f64	ms	per-req
11	inference_throughput	f64	req/s	1 Hz
12	error_rate	f64	errors/s	1 Hz

3.3 Manifest File and Cross-Ledger Linking

The manifest file provides summary statistics and cross-references:

Algorithm 1: Health Manifest Generation

Input: Health ledger H, primary audit ledger A

Output: Manifest M (JSON)

```
1: M.timespan ← (H.first_entry.timestamp, H.last_entry.timestamp)
2: M.entry_count ← H.entry_count
3: M.health_head_hash ← H.chain_head
4: M.primary_head_hash ← A.chain_head (if available)
5: M.cross_hash ← SHA3-256(H.chain_head || A.chain_head)
6: M.metric_summary ← for each metric_id in H:
7:   { id, min, max, avg, count, last }
8: M.alerts ← H.filter(severity >= 2)
9: M.signature ← Ed25519_Sign(k_manifest, SHA3-256(M))
10: return M
```

3.4 Retention and Pruning

Health ledgers implement configurable retention pruning:

Algorithm 2: Health Ledger Pruning

Input: Health ledger H, retention window R (e.g., 90 days)

Output: Pruned ledger H'

```
1: cutoff ← CurrentTime() - R
2: first_valid_index ←
3:   H.FindFirstEntry(timestamp >= cutoff)
4: if first_valid_index <= 0 then
```

```

5:   return H // Nothing to prune
6: end if
7: H' ← CreateEmptyHealthLedger()
8: H'.chain_head ← H[first_valid_index].parent_hash
9:   // External anchor: must verify against manifest
10: for i in first_valid_index..H.last_index do
11:   H'.Append(H[i])
12: end for
13: return H'

```

Pruning preserves hash chain integrity from the first retained entry. The manifest provides the anchor for the pruned chain.

3.5 Performance Overhead

Benchmark configuration: AMD EPYC 7702, 256 GB RAM, Ubuntu 22.04.

Health monitoring overhead (1 Hz polling):

Metric	Value
CPU utilization	0.3% of 1 core
Memory consumption	48 MB
Storage (raw health)	1.8 MB/hour
Storage (with manifest)	2.1 MB/hour
Append latency (p99)	1.2 ms
Integrity verification	0.8 s / 1M entries

Comparison of integration approaches:

Approach	CPU Overhead	Storage/Hour	Verify Time	Isolation
Integrated (primary)	0.4%	3.5 MB	1.2 s	None
Parallel (health)	0.3%	2.1 MB	0.8 s	Full
External (syslog)	0.1%	0.8 MB	N/A	Full

3.6 Health Alert Detection

The health ledger supports programmable alert rules:

```

Rule: AI model performance degradation
Condition: inference_latency > 10s over 5 consecutive entries
Action: log critical health entry
        notify operations team via webhook
        cross-reference to primary audit ledger

```

Algorithm 3: Health Alert Rule Evaluation

Input: New health entry e , rule set R

Output: Triggered alerts A

```

1: A ← empty list
2: for each rule r in R do
3:   if e.metric_group = r.metric_group then
4:     history ← GetRecentMetrics(r.metric_id, r.window)
5:     if EvaluateRule(r, history) = true then
6:       alert ← CreateAlert(r, e, history)
7:       A.add(alert)
8:     end if
9:   end if
10: end for
11: return A

```

4. Current State of the Art

4.1 Alternative Health Monitoring Approaches

Prometheus + Grafana: Standard for metrics collection and visualization. No cryptographic integrity. Uses pull model with time-series database (TSDB). Retention through downsampling.

Datadog/New Relic: Commercial observability platforms with rich dashboards and alerting. Health data stored in proprietary format without user-verifiable integrity.

Collectd/Telegraf: Agent-based metrics collection with multiple output plugins. Can write to text files but not to integrity-protected ledgers.

Linux Auditd: Kernel-level auditing with immutable log configuration. Limited to system calls rather than health metrics. Hash chain integrity requires external tooling.

4.2 AI-Specific Monitoring

MLflow Model Registry: Tracks model versions and deployment stages but not runtime health.

WhyLabs AI Observability provides model performance monitoring with data drift detection but without cryptographic integrity. **Seldon Core** provides model deployment monitoring with Prometheus metrics export.

5. Relevance to AIOSS

5.1 AIOSS Health Ledger CLI

```

aiossh health init --output system_health.aiosh
aiossh health monitor --health system_health.aiosh \
    --interval 1s --metrics all
aiossh health verify --health system_health.aiosh
aiossh health manifest --health system_health.aiosh \
    --primary audit.aioss \
    --output manifest.json
aiossh health prune --health system_health.aiosh \
    --retention 90d
aiossh health export --health system_health.aiosh \
    --format csv --output health_export.csv

```

5.2 Regulatory Compliance

- **EU AI Act Article 14:** High-risk AI systems must maintain “technically robust and accurate” logs. Health ledgers provide evidence of system reliability.
- **ISO 27001 Annex A.12.4:** Monitoring and measurement of system performance requires integrity-protected logs.
- **HIPAA Security Rule §164.312(b):** Audit controls extend to system monitoring activities.

5.3 Forensics Integration

In incident response scenarios, the health ledger provides time-series evidence of system behavior preceding and during an incident. Cross-referencing with the primary audit ledger enables correlated forensic analysis:

1. Identify anomaly onset in health metrics (CPU spike, latency increase)
2. Correlate with primary audit events (model inference requests, data access)
3. Verify integrity of both ledgers through manifest cross-hash
4. Generate forensics report with cryptographic proof of evidence preservation

6. Future Directions

6.1 Adaptive Sampling Rate

Health monitoring at fixed intervals wastes storage during quiescent periods and undersamples during incidents. Adaptive sampling (Yin et al., 2019) adjusts polling frequency based on metric volatility. For AI workloads, inference latency variance could trigger increased sampling during periods of instability.

6.2 Predictive Health Analytics

Machine learning models trained on health ledger history could predict incipient failures (Lou et al., 2022). AI-powered root cause analysis using health ledger time-series data could automate incident diagnosis.

6.3 Edge Device Health Ledgers

For AI systems deployed on edge devices with constrained storage, health ledger pruning must be more aggressive. Distributed health ledger architectures where edge devices push periodic health summaries to a central ledger (Wang et al., 2023) could provide integrity at scale.

Works Cited

1. Case, J., Fedor, M., Schoffstall, M., & Davin, J. (1990). Simple Network Management Protocol (SNMP). *RFC 1157*. <https://doi.org/10.17487/RFC1157>
2. Turnbull, J. (2018). *Monitoring with Prometheus*. Turnbull Press.
3. CNCF. (2024). OpenTelemetry Specification. *Cloud Native Computing Foundation*. <https://opentelemetry.io/docs/specs/otel/>

4. Baylor, D., Breck, E., Cheng, H.-T., Fiedel, N., Foo, C. Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., Kuo, C. Y., Lew, L., Minkus, S., Nemirovsky, D., Park, N., Rao, V., Shafran, I., Stewart, M., Tang, D., ... & Yoo, G. (2017). TFX: A TensorFlow-based production-scale machine learning platform. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1387–1395. <https://doi.org/10.1145/3097983.3098021>
5. Hermann, J., & Del Balso, M. (2017). Meet Michelangelo: Uber’s machine learning platform. *Uber Engineering Blog*. <https://eng.uber.com/michelangelo/>
6. Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., Xie, F., & Zumar, C. (2018). Accelerating the machine learning lifecycle with MLflow. *IEEE Data Engineering Bulletin*, 41(4), 39–45.
7. Crosby, S. A., & Wallach, D. S. (2009). Efficient data structures for tamper-evident logging. *Proceedings of the 18th USENIX Security Symposium*, 317–334.
8. Pullkis, M., Surak, A., & Znotins, A. (2020). Performance analysis of hash chain verification in cloud environments. *IEEE Access*, 8, 112455–112471. <https://doi.org/10.1109/ACCESS.2020.3003156>
9. Hadosmanović, D., Simionato, L., & Wiemers, N. (2015). Secure audit logging for industrial control systems. *International Conference on Critical Information Infrastructures Security*, 145–158. https://doi.org/10.1007/978-3-319-31664-2_12
10. Dorri, A., Kanhere, S. S., Jurdak, R., & Gauravaram, P. (2017). Blockchain for IoT security and privacy: The case study of a smart home. *IEEE International Conference on Pervasive Computing and Communications Workshops*, 618–623. <https://doi.org/10.1109/PERCOMW.2017.7917634>
11. Yue, X., Wang, H., Jin, D., Li, M., & Jiang, W. (2016). Healthcare data gateways: Found healthcare intelligence on blockchain with novel privacy risk control. *Journal of Medical Systems*, 40, 218. <https://doi.org/10.1007/s10916-016-0574-6>
12. Grubb, S. (2004). Linux Audit Subsystem. *Red Hat*. <https://people.redhat.com/sgrubb/audit/>
13. Yin, X., Yang, H., & Wang, J. (2019). Adaptive sampling for time-series monitoring with bounded error. *ACM SIGMOD International Conference on Management of Data*, 1567–1584. <https://doi.org/10.1145/3299869.3319892>
14. Lou, Y., Fu, Q., Yang, S., & Li, J. (2022). Predictive analytics for cloud system failures using time-series models. *IEEE Transactions on Cloud Computing*, 10(3), 1821–1835. <https://doi.org/10.1109/TCC.2020.2998478>
15. Wang, Z., Liu, B., & Zhang, Y. (2023). Distributed health ledgers for edge AI deployments. *IEEE Internet of Things Journal*, 10(8), 6897–6912. <https://doi.org/10.1109/JIOT.2022.3226455>
16. Baysal, O., Holmes, R., & Godfrey, M. W. (2013). Developer dashboards: The need for qualitative analytics. *IEEE Software*, 30(4), 46–52. <https://doi.org/10.1109/MS.2013.54>
17. Shang, W., Jiang, Z. M., Hemmati, H., Adams, B., Hassan, A. E., & Martin, P. (2015). Assisting developers of big data analytics applications when deploying on Hadoop clouds. *IEEE Transactions on Software Engineering*, 41(5), 442–466. <https://doi.org/10.1109/TSE.2014.2383384>
18. He, S., Zhu, J., He, P., & Lyu, M. R. (2016). Experience report: System log analysis for anomaly detection. *IEEE International Symposium on Software Reliability Engineering*, 207–218. <https://doi.org/10.1109/ISSRE.2016.21>

19. Lin, Q., Lou, J., Zhang, H., & Zhang, D. (2019). Log clustering for problem determination in large-scale service systems. *IEEE Transactions on Services Computing*, 12(3), 423–436. <https://doi.org/10.1109/TSC.2016.2632700>
20. Fu, Q., Lou, J.-G., Wang, Y., & Li, J. (2009). Execution anomaly detection in distributed systems through unstructured log analysis. *IEEE International Conference on Data Mining*, 129–140. <https://doi.org/10.1109/ICDM.2009.60>
21. Wei, J., Zhang, Z., & Zhang, C. (2022). Performance anomaly detection in cloud systems using deep learning on health metrics. *IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*, 145–154. <https://doi.org/10.1109/CCGrid55645.2022.00023>
22. Abad, C. L., & Bonilla, R. M. (2021). Anomaly detection in AI inference pipelines. *IEEE International Conference on Big Data*, 2345–2354. <https://doi.org/10.1109/BigData52589.2021.00087>
23. Bao, L., Liu, X., & Chen, W. (2020). Learning-based resource monitoring for cloud-native AI systems. *IEEE Transactions on Parallel and Distributed Systems*, 31(12), 2832–2846. <https://doi.org/10.1109/TPDS.2020.3006782>
24. Agrawal, S., & Agrawal, J. (2015). Survey on anomaly detection using data mining techniques. *Procedia Computer Science*, 60, 708–713. <https://doi.org/10.1016/j.procs.2015.08.220>
25. Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 41(3), 1–58. <https://doi.org/10.1145/1541880.1541882>
26. Hodge, V. J., & Austin, J. (2004). A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22, 85–126. <https://doi.org/10.1023/B:AIRE.0000045502.10941.a9>
27. Hawkins, D. M. (1980). *Identification of Outliers*. Springer. <https://doi.org/10.1007/978-94-015-3994-4>
28. Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). LOF: Identifying density-based local outliers. *ACM SIGMOD International Conference on Management of Data*, 93–104. <https://doi.org/10.1145/342009.335388>
29. Aggarwal, C. C. (2017). *Outlier Analysis* (2nd ed.). Springer. <https://doi.org/10.1007/978-3-319-47578-3>
30. Zhuang, Y., & Chen, L. (2021). Health monitoring for AI systems: Metrics and challenges. *IEEE Software*, 38(2), 78–86. <https://doi.org/10.1109/MS.2020.3033321>
31. Chen, L., & Zhuang, Y. (2022). Integrity-preserving system monitoring for regulated AI deployments. *IEEE Transactions on Dependable and Secure Computing*, 19(4), 2456–2471. <https://doi.org/10.1109/TDSC.2021.3067821>
32. Shapiro, J. S. (2020). Health monitoring architecture for mission-critical AI systems. *ACM Workshop on AI System Monitoring*, 23–32. <https://doi.org/10.1145/3423409.3423415>
33. European Commission. (2024). Regulation (EU) 2024/1689 (EU AI Act). *Official Journal of the European Union*. <https://eur-lex.europa.eu/eli/reg/2024/1689/oj>
34. ISO/IEC. (2022). ISO/IEC 27001:2022 Information security management systems. <https://www.iso.org/standard/82875.html>
35. HHS. (2013). HIPAA Administrative Simplification Regulation. *45 CFR Parts 160, 162, and 164*. <https://www.hhs.gov/hipaa/for-professionals/security/index.html>

36. AICPA. (2020). SOC 2 trust services criteria. *American Institute of CPAs*. <https://www.aicpa.org/trustservices>
37. GSA. (2022). FedRAMP: Federal Risk and Authorization Management Program. <https://www.fedramp.gov/>
38. NIST. (2020). NIST SP 800-53 Rev. 5: Security and Privacy Controls for Information Systems and Organizations. <https://doi.org/10.6028/NIST.SP.800-53r5>
39. National Institute of Standards and Technology. (2015). SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. *FIPS PUB 202*. <https://doi.org/10.6028/NIST.FIPS.202>
40. Josefsson, S., & Liusvaara, I. (2017). Edwards-Curve Digital Signature Algorithm (EdDSA). *RFC 8032*. <https://doi.org/10.17487/RFC8032>
41. Dean, J., & Barroso, L. A. (2013). The tail at scale. *Communications of the ACM*, 56(2), 74–80. <https://doi.org/10.1145/2408776.2408794>
42. Barroso, L. A., Clidaras, J., & Hölzle, U. (2013). The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 8(3), 1–154. <https://doi.org/10.2200/S00516ED2V01Y201306CAC024>
43. Siegwart, D., & Harper, M. (2022). Efficient time-series storage for high-frequency metrics. *ACM Transactions on Storage*, 18(4), 1–27. <https://doi.org/10.1145/3543047>
44. Pelkonen, T., Franklin, S., Teller, J., Cavallaro, P., Huang, Q., Meza, J., & Veeraraghavan, K. (2015). Gorilla: A fast, scalable, in-memory time series database. *Proceedings of the VLDB Endowment*, 8(12), 1816–1827. <https://doi.org/10.14778/2824032.2824080>
45. Deri, L., & Martinelli, M. (2014). High-speed network monitoring with ntopng. *IEEE International Symposium on Integrated Network Management*, 1102–1103. <https://doi.org/10.1109/INM.2013.657310>
46. Jammal, M., Singh, T., Shami, A., Asal, R., & Li, Y. (2014). Software defined networking: State of the art and research challenges. *Computer Networks*, 72, 74–98. <https://doi.org/10.1016/j.comnet.2014.07.004>
47. Hellerstein, J. M., Stonebraker, M., & Hamilton, J. R. (2007). Architecture of a database system. *Foundations and Trends in Databases*, 1(2), 141–259. <https://doi.org/10.1561/19000000002>
48. Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., Bhagat, N., Mittal, S., & Ryaboy, D. (2014). Storm @Twitter. *ACM SIGMOD International Conference on Management of Data*, 147–156. <https://doi.org/10.1145/2588555.2595641>
49. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013). Discretized streams: Fault-tolerant streaming computation at scale. *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, 423–438. <https://doi.org/10.1145/2517349.2522737>
50. Stonebraker, M., Çetintemel, U., & Zdonik, S. (2005). The 8 requirements of real-time stream processing. *ACM SIGMOD Record*, 34(4), 42–47. <https://doi.org/10.1145/1107499.1107504>
51. Akidau, T., Schmidt, E., Whittle, R., Bradshaw, R., Chernyak, S., & Chernyak, S. (2015). The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8(12), 1792–1803. <https://doi.org/10.14778/2824032.2824076>

52. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache Flink: Stream and batch processing in a single engine. *IEEE Data Engineering Bulletin*, 38(4), 28–38.
53. Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A distributed messaging system for log processing. *Proceedings of the 6th International Workshop on Networking Meets Databases*, 1–7.
54. Neumeyer, L., Robbins, B., Nair, A., & Kesari, A. (2010). S4: Distributed stream computing platform. *IEEE International Conference on Data Mining Workshops*, 170–177. <https://doi.org/10.1109/ICDMW.2010.172>
55. Gedik, B., Andow, L., Surtani, A., & Mutchler, P. (2018). IBM Streams: A platform for analyzing big data in motion. *IBM Journal of Research and Development*, 62(6), 1–13. <https://doi.org/10.1147/JRD.2018.2874160>

Lois-Kleinner & 0-1.gg 2026 — AIOSS (AI Open Signed Storage)